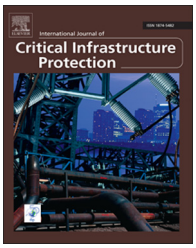
Available online at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ijcip



A methodology for determining the image base of ARM-based industrial control system firmware

Ruijin Zhu[a,b], *Baofeng Zhang*[a], *Junjie Mao*[a], *Quanxin Zhang*[b,c], *Yu-an Tan*[b,c,*]

[a]*China Information Technology Security Evaluation Center, Beijing 100085, China*
[b]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China*
[c]*Beijing Engineering Research Center for Massive Language Information Processing and Cloud Computing Applications, Beijing 100081, China*

ARTICLE INFO

Article history:
Received 1 January 2016
Received in revised form
30 July 2016
Accepted 28 November 2016
Available online 3 January 2017

Keywords:
Industrial Control Systems
ARM Architecture
Firmware
Image Base
Reverse Engineering

ABSTRACT

A common way to evaluate the security of an industrial control system is to reverse engineer its firmware; this is typically performed when the source code of the device is not available and the firmware is not trusted. However, many industrial control systems are based on the ARM architecture for which the firmware format is always unknown. Therefore, it is difficult to obtain the image base of firmware directly, which significantly complicates reverse engineering efforts. This paper describes a methodology for automatically determining the image base of firmware of ARM-based industrial control systems. Two algorithms, FIND-String and FIND-LDR, are presented that obtain the offsets of strings in firmware and the string addresses loaded by LDR instructions, respectively. Additionally, the DBMSSL algorithm is presented that uses the outputs of the FIND-String and FIND-LDR algorithms to determine the image base of firmware. Experiments are performed with 10 samples of industrial control system firmware collected from the Internet. The experimental results demonstrate that the proposed methodology is effective at determining the image bases of the majority of the firmware samples.

1. Introduction

Industrial control systems are widely used in critical infrastructure assets such as water treatment plants, oil and gas pipelines, refineries and electric power grids. Traditionally, industrial control systems have been designed for operation in closed, trusted networks with little emphasis on security and limited protection mechanisms [5]. However, increased interconnectivity, especially connections to corporate networks and the Internet expose industrial control systems and the critical infrastructures they monitor and control to serious threats.

One example is Stuxnet, which, in 2010, targeted uranium hexafluoride centrifuges at Natanz in Iran [10]. In 2011, SCADA systems at water utilities in Illinois were hacked, which disrupted the water supply [8]. In 2014, the U.S. ICS-CERT [9] released a security bulletin about the Havex malware. Like Stuxnet, Havex was designed to attack industrial control systems; it supposedly has the ability to disable hydropower dams, overload nuclear power plants and even shut down power grids.

Statistics indicate that 92.6% of the vulnerabilities discovered in current industrial control systems are in software/firmware

*Corresponding author at: School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.
E-mail address: tan2008@bit.edu.cn (Y.-a. Tan).

http://dx.doi.org/10.1016/j.ijcip.2016.12.002
1874-5482/

whereas only 7.4% are associated with hardware [11]. Any firmware used in industrial control systems should be assumed to be insecure because it may contain vulnerabilities and security flaws. Therefore, it is imperative to conduct security analyses and vulnerability discovery efforts for industrial control systems [12,13,18,20].

The security of firmware can be analyzed by reverse engineering [3,13,14,19,21]. When disassembling firmware, a tool such as IDA Pro needs to know the processor type and image base. In general, the processor type can be discerned by consulting the product manual or tearing down the device. If the firmware format is known, then the image base can be discerned. Unfortunately, most ARM firmware, which is widely used in modern industrial control systems, are binary files with unknown formats, so it is difficult to obtain the image bases directly. Armed with the correct image base, a disassembler can construct accurate cross references in instances where the address references use absolute addresses instead of offsets in a binary file [16]. The cross references, which include jump location references, function references, string references, etc., can be very helpful when attempting to navigate messy disassembled code. Hence, identifying the image base is important for reverse engineering efforts.

Several solutions have been proposed to obtain the image base of firmware with an unknown format. Skochinsky [17] has proposed a general technique for determining the image base of embedded system firmware; the technique leverages several hints such as self-relocating code and initialization code. Basnight et al. [2,4] have presented two methods for inferring an image base. The first method uses immediate values in firmware instruction and update files to infer a reasonable image base. The second method uses a hardware debugger to connect to and halt a programmable logic controller and obtain a memory dump. The image base is then found by manually analyzing common ARM instruction patterns in the memory dump.

Da Costa et al. [7] have noted that, when the case values in a `switch` statement of a C program are sequential and dense, the memory addresses of the cases are usually stored in a jump table; this fact can be used to infer the memory addresses of pieces of nearby code and eventually obtain the image base. Santamarta [15] describes another way to use a jump table. Since a jump table contains the absolute addresses of cases, the distances between the cases can be calculated. If there is a certain distance that is different from the others, the corresponding relation between the absolute address and offset of the case can be obtained, based on which, the base address can be determined.

These solutions for determining an image base need human interaction, namely the determination relies on the intuition and experience of the reverse engineer. Analysis of the literature reveals that only the methods described in [22,23] can automatically calculate the image base of firmware with an unknown format. However, the efficiency of these methods needs to be improved.

At present, most industrial control system firmware resides in embedded systems. According to Costin et al. [6], approximately 63% of embedded devices are based on the ARM architecture. Hence, this research focuses on industrial systems based on the ARM architecture and proposes a methodology for determining their firmware image bases. Firmware usually contains strings and the strings that are referenced in adjacent code are stored centrally. Therefore, to begin with, the FIND-String algorithm is presented for obtaining the string offsets used to calculate the numbers of bytes occupied by the strings. Since a compiler typically loads a string address into a register using the LDR instruction, the characteristics of the LDR encoding format are leveraged to specify the FIND-LDR algorithm that obtains the addresses of the strings loaded by LDR instructions. Next, the number of bytes occupied by the strings are calculated. Finally, using the numbers of bytes occupied by strings provided by the FIND-String and FIND-LDR algorithms, it is possible to discern the relationships between string offsets and memory addresses, which yield the image base of firmware.

This research has two main contributions. First, it work leverages the encoding of LDR instructions to effectively identify LDR instructions and calculate the address loaded by each LDR instruction. Second, a methodology is presented for determining the image base of industrial control system firmware with an unknown format. The methodology uses string offsets and string addresses loaded by LDR instructions to determine the image base. Experiments demonstrate that the methodology is very effective at determining the image base of firmware that uses LDR instructions to load string addresses.

2. Strings and LDR instructions in firmware

This section discusses the storage features and loading process of strings in firmware. The FIND-String algorithm is presented for recognizing strings and outputting their offsets. Additionally, the FIND-LDR algorithm is presented for identifying LDR instructions in firmware and outputting the addresses loaded by LDR instructions.

2.1. *Identifying strings in firmware*

A binary file typically contains a number of strings, including prompt messages, error messages and version information. Each string contains some printable characters and escape characters. Printable characters include letters, numbers and punctuation; the ASCII range of these characters is `0x20` to `0x7E`. Escape characters include line breaks (`0x0a`), tabs (`0x09`) and others; the ASCII range of these characters is `0x09` to `0x0D`. Thus, the ASCII range of strings is [`0x09`, `0x0D`] ∪[`0x20`, `0x7E`]. Since the C language is most commonly used for developing industrial control system software, only C-style strings are discussed in this paper. In the C language, a string is usually stored in a character array whose last element is the string terminator "`\0''` with ASCII code `0x00`.

Figs. 1(a) and 1(b) show strings stored in the compact mode and aligned mode, respectively. In both storage modes, the strings are stored by the compiler. For performance reasons, some compilers store strings in the aligned mode. If the available storage position for a string is not a multiple of four bytes, the compiler adds some padding characters (`0x00` bytes) to create a storage position that is an exact